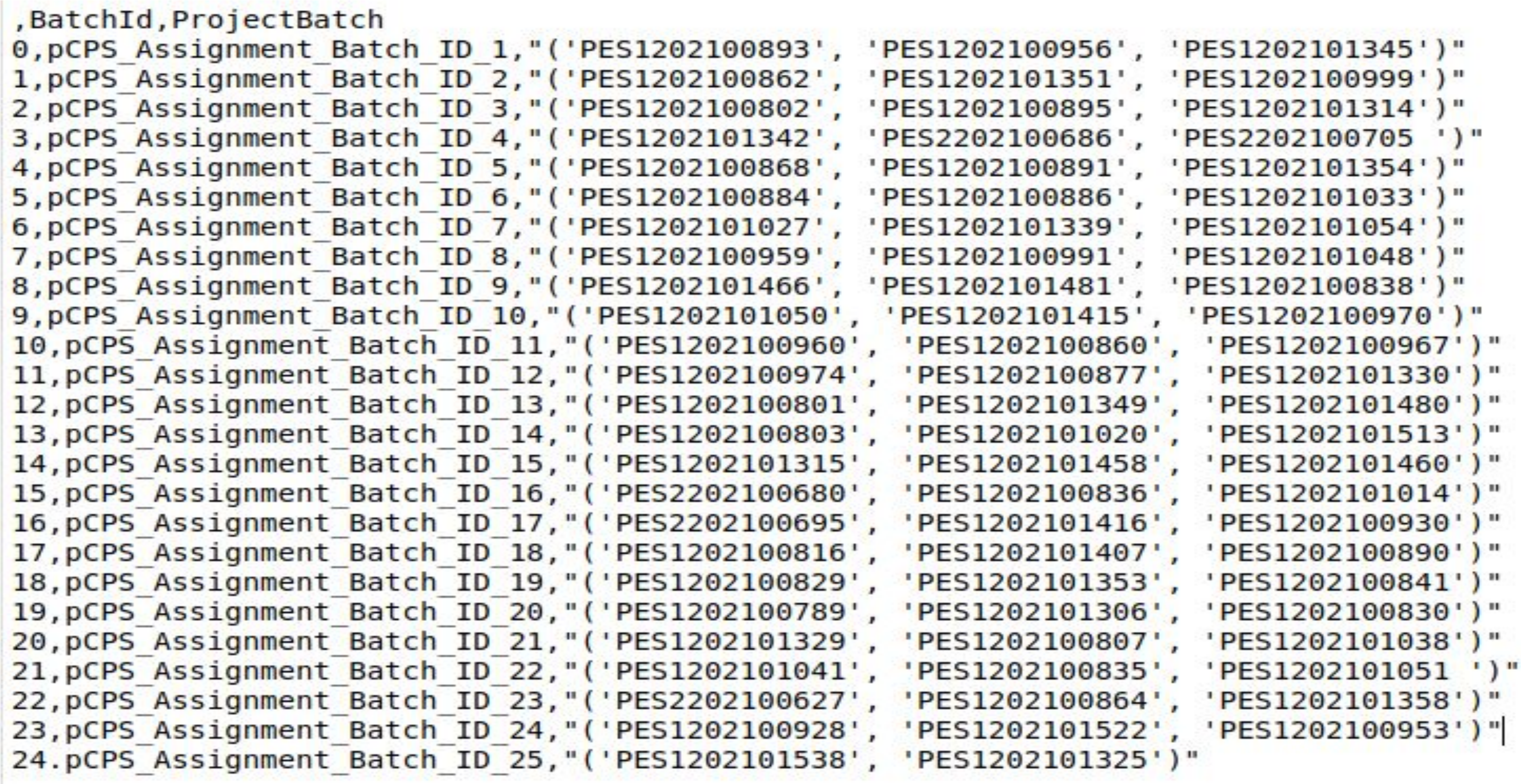
python for Computational Problem Solving - pCPS - Lists Lecture Slides - Class #17_#18

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

pCPS Assignment Batches



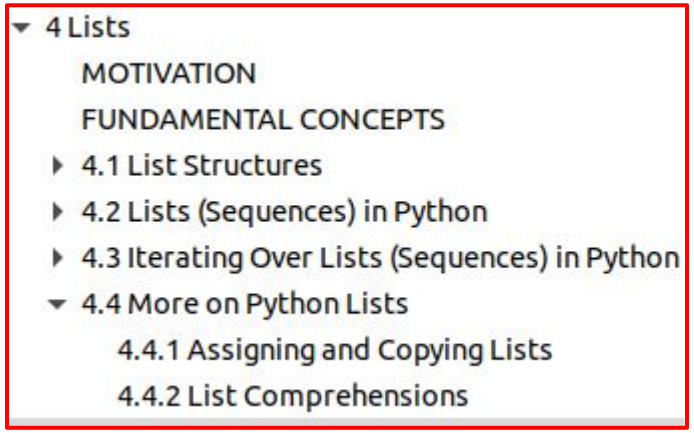
```
,BatchId,ProjectBatch
0,pCPS_Assignment_Batch_ID_1,"('PES1202100893', 'PES1202100956', 'PES1202101345')"
1,pCPS_Assignment_Batch_ID_2,"('PES1202100862', 'PES1202101351', 'PES1202100999')"
2,pCPS_Assignment_Batch_ID_3,"('PES1202100802', 'PES1202100895', 'PES1202101314')"
3,pCPS_Assignment_Batch_ID_4,"('PES1202101342', 'PES2202100686', 'PES2202100705 ')"
4,pCPS_Assignment_Batch_ID_5,"('PES1202100868', 'PES1202100891', 'PES1202101354')"
5,pCPS_Assignment_Batch_ID_6,"('PES1202100884', 'PES1202100886', 'PES1202101033')"
6,pCPS_Assignment_Batch_ID_7,"('PES1202101027', 'PES1202101339', 'PES1202101054')"
7,pCPS_Assignment_Batch_ID_8,"('PES1202100959', 'PES1202100991', 'PES1202101048')"
8,pCPS_Assignment_Batch_ID_9,"('PES1202101466', 'PES1202101481', 'PES1202100838')"
9,pCPS_Assignment_Batch_ID_10,"('PES1202101050', 'PES1202101415', 'PES1202100970')"
10,pCPS_Assignment_Batch_ID_11,"('PES1202100960', 'PES1202100860', 'PES1202100967')"
11,pCPS_Assignment_Batch_ID_12,"('PES1202100974', 'PES1202100877', 'PES1202101330')"
12,pCPS_Assignment_Batch_ID_13,"('PES1202100801', 'PES1202101349', 'PES1202101480')"
13,pCPS_Assignment_Batch_ID_14,"('PES1202100803', 'PES1202101020', 'PES1202101513')"
14,pCPS_Assignment_Batch_ID_15,"('PES1202101315', 'PES1202101458', 'PES1202101460')"
15,pCPS_Assignment_Batch_ID_16,"('PES2202100680', 'PES1202100836', 'PES1202101014')"
16,pCPS_Assignment_Batch_ID_17,"('PES2202100695', 'PES1202101416', 'PES1202100930')"
17,pCPS_Assignment_Batch_ID_18,"('PES1202100816', 'PES1202101407', 'PES1202100890')"
18,pCPS_Assignment_Batch_ID_19,"('PES1202100829', 'PES1202101353', 'PES1202100841')"
19,pCPS_Assignment_Batch_ID_20,"('PES1202100789', 'PES1202101306', 'PES1202100830')"
20,pCPS_Assignment_Batch_ID_21,"('PES1202101329', 'PES1202100807', 'PES1202101038')"
21,pCPS_Assignment_Batch_ID_22,"('PES1202101041', 'PES1202100835', 'PES1202101051 ')"
22,pCPS_Assignment_Batch_ID_23,"('PES2202100627', 'PES1202100864', 'PES1202101358')"
23,pCPS_Assignment_Batch_ID_24,"('PES1202100928', 'PES1202101522', 'PES1202100953')"
24.pCPS_Assignment_Batch_ID_25,"('PES1202101538', 'PES1202101325')"
```

python for Computational Problem Solving Syllabus

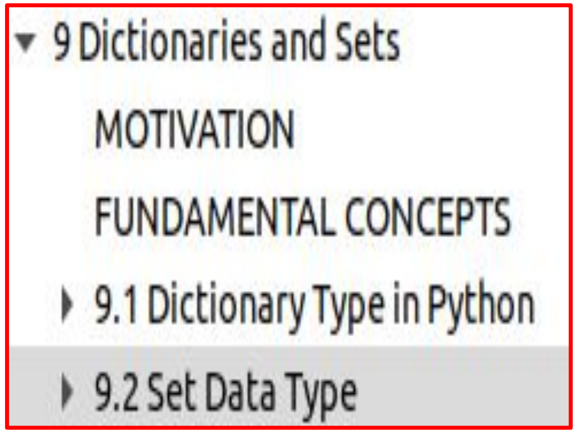
Unit II: Collections & Basics of Functions - 12 Hours

Lists, Tuples , Dictionaries, Sets, Strings and text file manipulation: reading and writing files. Functions : Definition, call.

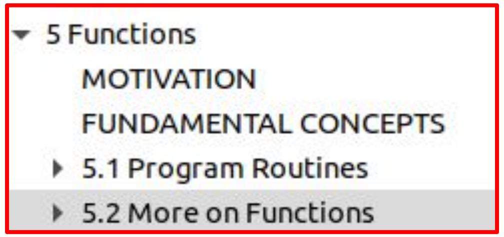
T1: 4.1 – 4.4 - Class #15, #16, #17, #18
T1: 9.1 – 9.2 - Class #19, #20, #21, #22
T1: 5.1-5.2 - Class #23, #24
T1: 8.1, 8.2, 8.3 - Class #25, #26



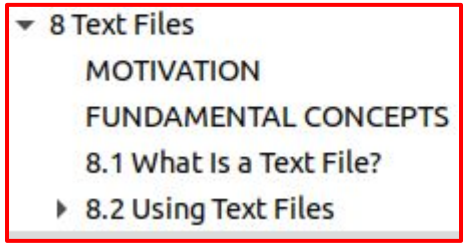
- ▾ 4 Lists
  - MOTIVATION
  - FUNDAMENTAL CONCEPTS
  - ▸ 4.1 List Structures
  - ▸ 4.2 Lists (Sequences) in Python
  - ▸ 4.3 Iterating Over Lists (Sequences) in Python
  - ▾ 4.4 More on Python Lists
    - 4.4.1 Assigning and Copying Lists
    - 4.4.2 List Comprehensions



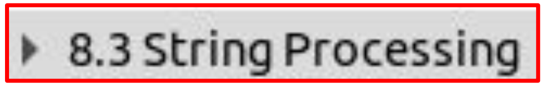
- ▾ 9 Dictionaries and Sets
  - MOTIVATION
  - FUNDAMENTAL CONCEPTS
  - ▸ 9.1 Dictionary Type in Python
  - ▸ 9.2 Set Data Type



- ▾ 5 Functions
  - MOTIVATION
  - FUNDAMENTAL CONCEPTS
  - ▸ 5.1 Program Routines
  - ▸ 5.2 More on Functions

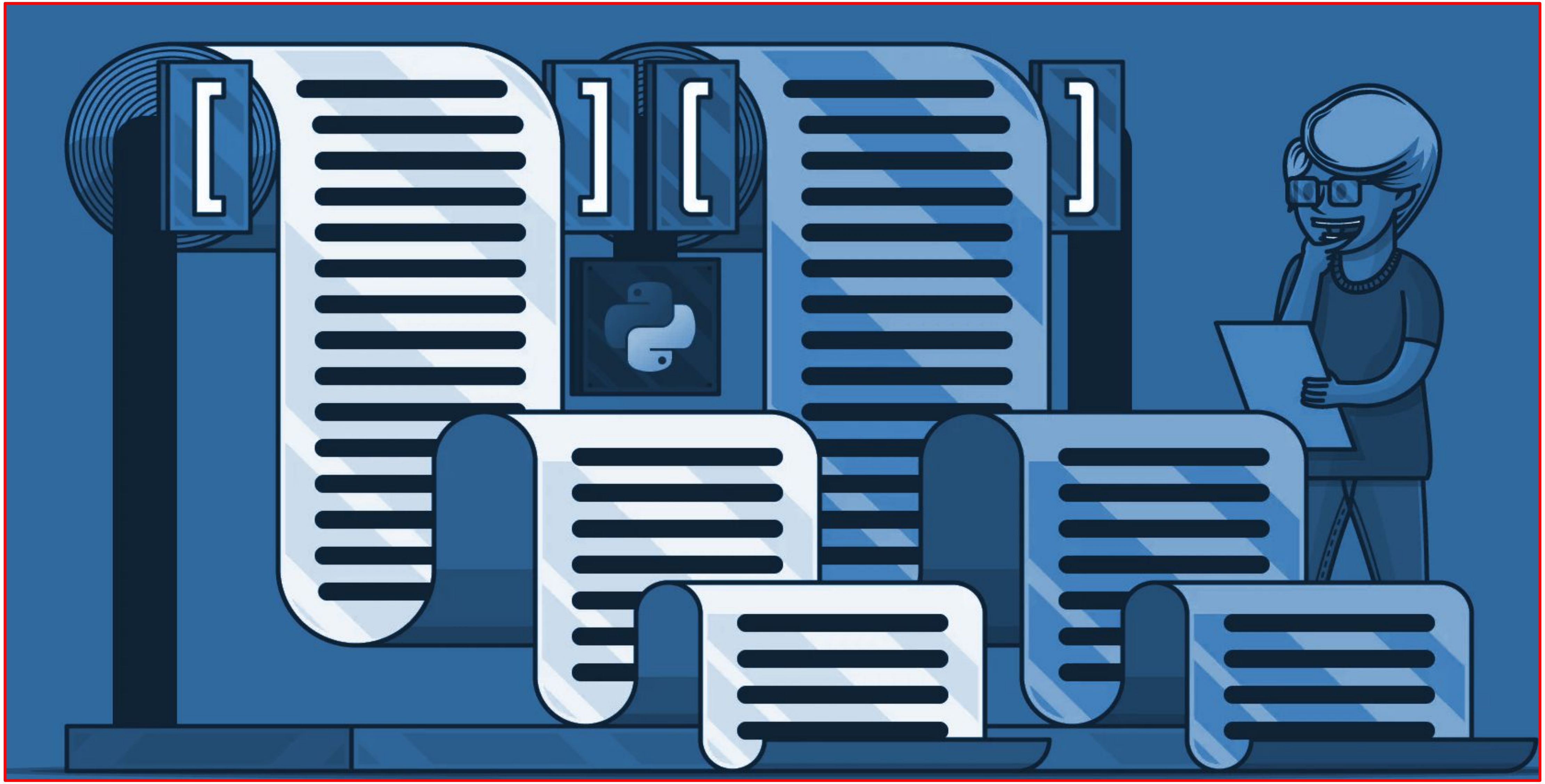


- ▾ 8 Text Files
  - MOTIVATION
  - FUNDAMENTAL CONCEPTS
  - 8.1 What Is a Text File?
  - ▸ 8.2 Using Text Files



- ▸ 8.3 String Processing

pCPS 4 Lists



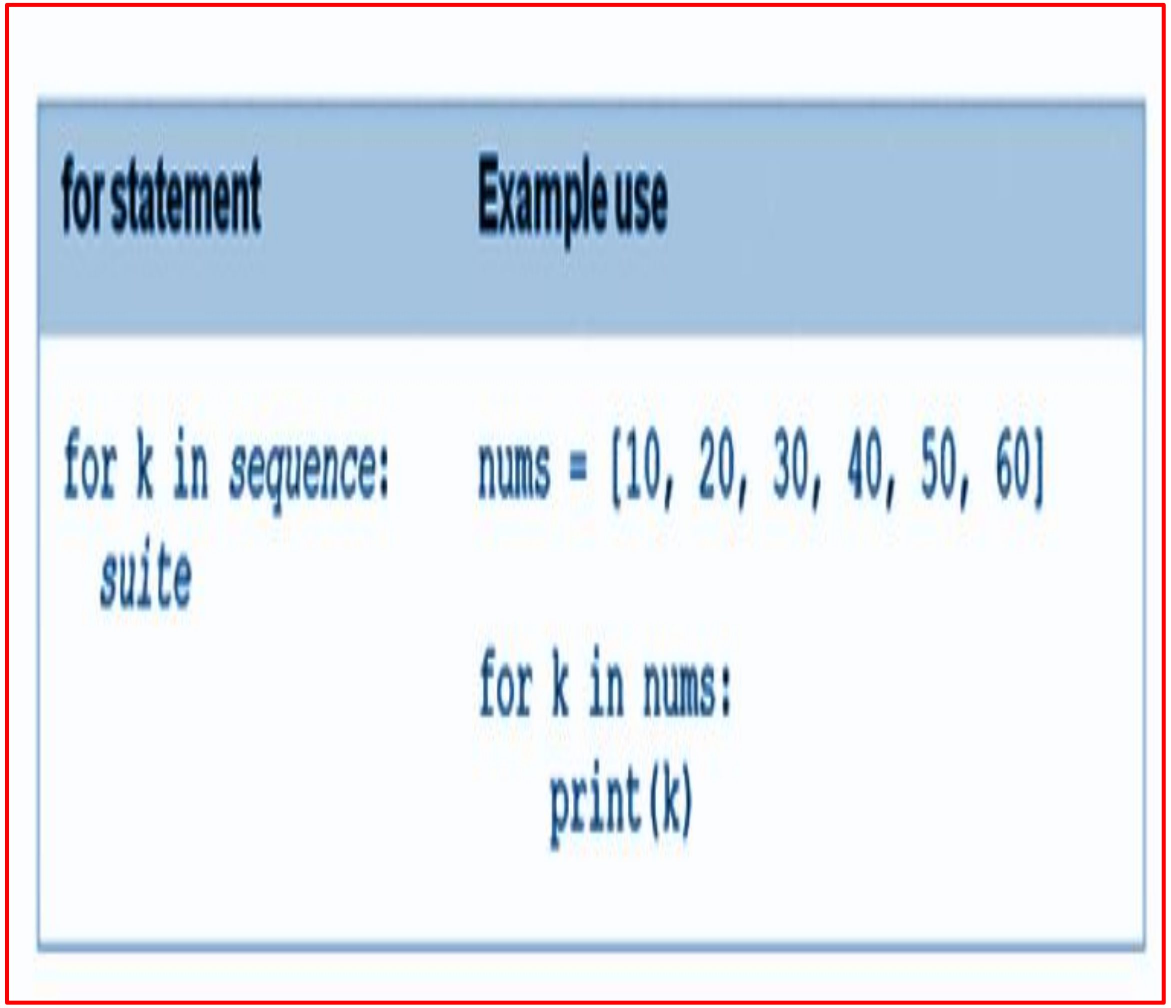
pCPS 4.3 Iterating Over Lists (Sequences) in python

- **Iteration** means the process of doing something again and again
- python's for statement provides a convenient means of **iterating** over lists and other sequences
- For list **iteration**, we look at both
  - **for loops**
  - **while loops**

pCPS 4.3.1 Loops in python

- A **for** statement is an **iterative control statement** that **iterates** once for **each** element in a **specified sequence** of **elements**.

- Variable **k** is referred to as a **loop** variable. Since there are **six** elements in the provided list, the for loop iterates exactly **six times**.



for statement	Example use

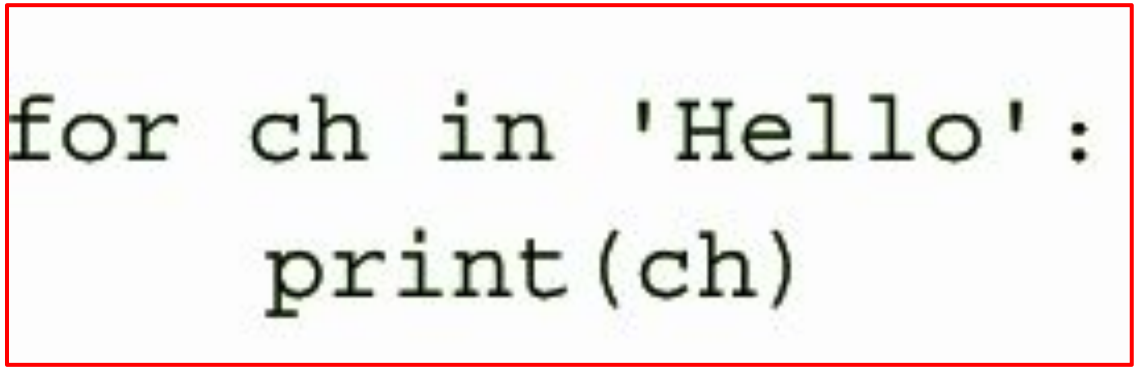
for k in sequence:
  suite
	nums = [10, 20, 30, 40, 50, 60]
	
	for k in nums:
	    print(k)

pCPS 4.3.1 Loops in python

- A **for** statement is an **iterative control statement** that **iterates** once for **each** element in a **specified sequence** of **elements**.

- The **for** statement can be applied to **all sequence** types, including **strings**.

- **Iteration** over a **string** can be done as shown

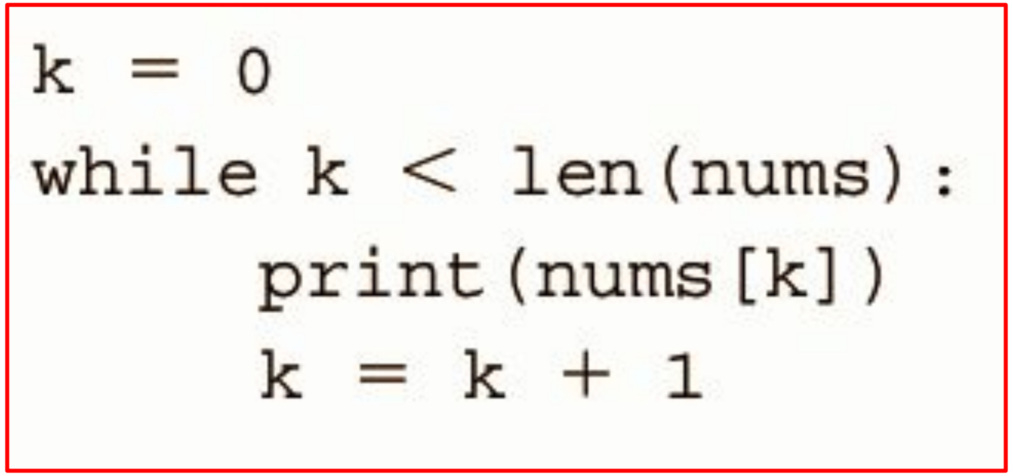


```
for ch in 'Hello':
    print(ch)
```

pCPS 4.3.1 Loops in python

- In the **while** loop version, loop variable **k** must be initialized to **0** and **incremented** by **1** **each** time through the loop.

- In the **for** loop version, loop variable **k** **automatically** iterates over the provided sequence of values

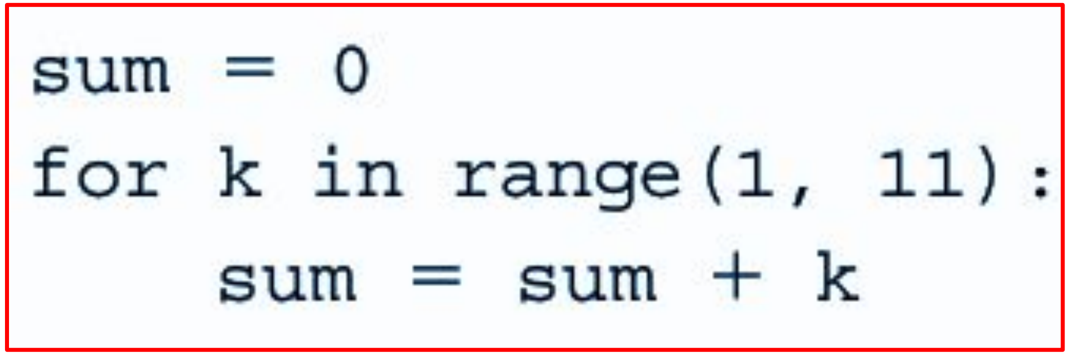


```
k = 0
while k < len(nums):
    print(nums[k])
    k = k + 1
```

pCPS 4.3.2 The Built-in range Function

- **python** provides a built-in **range** function that can be used for **generating** a sequence of **integers** that a **for** loop can **iterate** over

- The **values** in the **generated sequence** include the **starting value**, up to but **not including** the **ending value**.



```
sum = 0
for k in range(1, 11):
    sum = sum + k
```

pCPS 4.3.2 The Built-in range Function

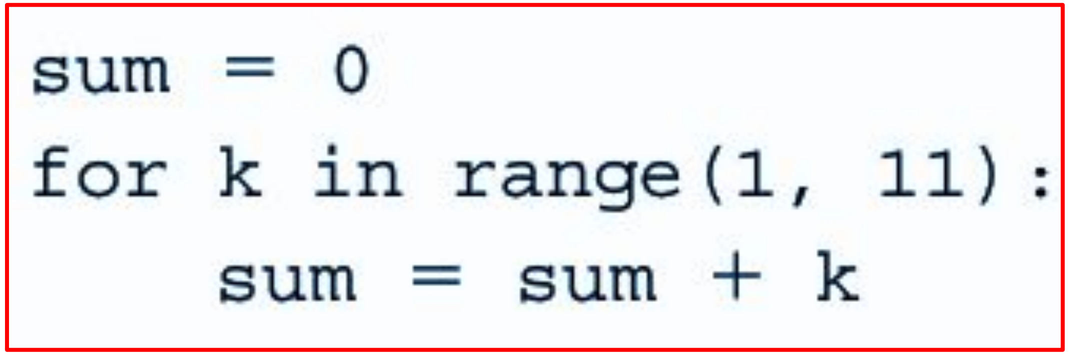
- The **range** function is convenient when long sequences of integers are needed.

- Actually, **range** does not create a sequence of integers.

- It **creates** a **generator** function able to **produce** each next item of the sequence when needed.

- This saves memory, especially for long lists.

- Therefore, typing **range(0, 9)** in the python shell **does** **not** produce a list as expected—it simply **“echoes out”** the call to range



```
sum = 0
for k in range(1, 11):
    sum = sum + k
```

pCPS 4.3.2 The Built-in range Function

- By default, the **range** function generates a sequence of consecutive integers.
- A “**step**” value can be provided.
- For example, range(0, 11, 2) produces the sequence [0, 2, 4, 6, 8, 10], with a step value of 2.
- A **sequence** can also be generated “**backwards**” when given a negative step value.
- For example, range(10, 0, -1) produces the sequence [10, 9, 8, 7, 6, 5, 4, 3, 2, 1].
- Note that since the generated sequence always begins with the provided starting value, “up to” but not including the final value

pCPS 4.3.2 The Built-in range Function

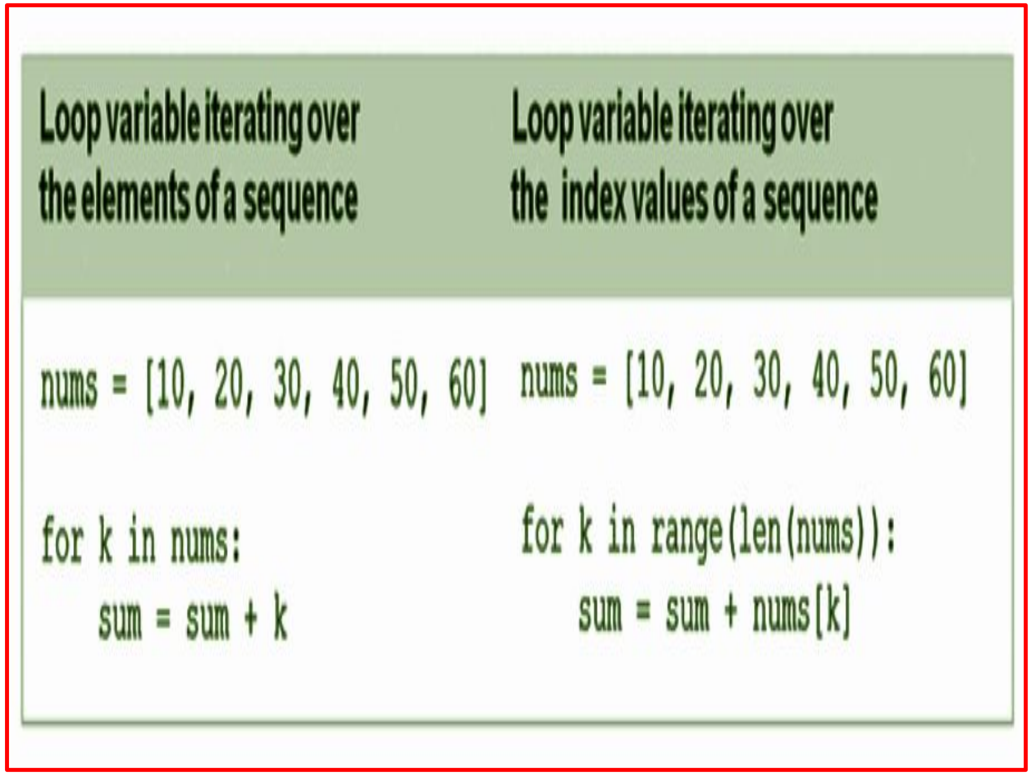
- By default, the **range** function generates a sequence of consecutive integers.
- A “**step**” value can be provided.
- For example, range(0, 11, 2) produces the sequence [0, 2, 4, 6, 8, 10], with a step value of 2.
- A **sequence** can also be generated “**backwards**” when given a negative step value.
- For example, range(10, 0, -1) produces the sequence [10, 9, 8, 7, 6, 5, 4, 3, 2, 1].
- Note that since the generated sequence always begins with the provided starting value, “up to” but not including the final value

pCPS 4.3.3 Iterating Over List Elements vs. List Index Values

- When the **elements** of a list need to be **accessed**, but **not** **altered**, a loop variable that iterates over each list element is an appropriate approach.

- There are times when the loop variable must **iterate** over the **index** **values** of a list instead

- An **index** **variable** is a variable whose **changing** **value** is used to **access** **elements** of an **indexed** data structure.



Loop variable iterating over the elements of a sequence	Loop variable iterating over the index values of a sequence
`nums = [10, 20, 30, 40, 50, 60]`	`nums = [10, 20, 30, 40, 50, 60]`
`for k in nums:`	`for k in range(len(nums)):`
`    sum = sum + k`	`    sum = sum + nums[k]`

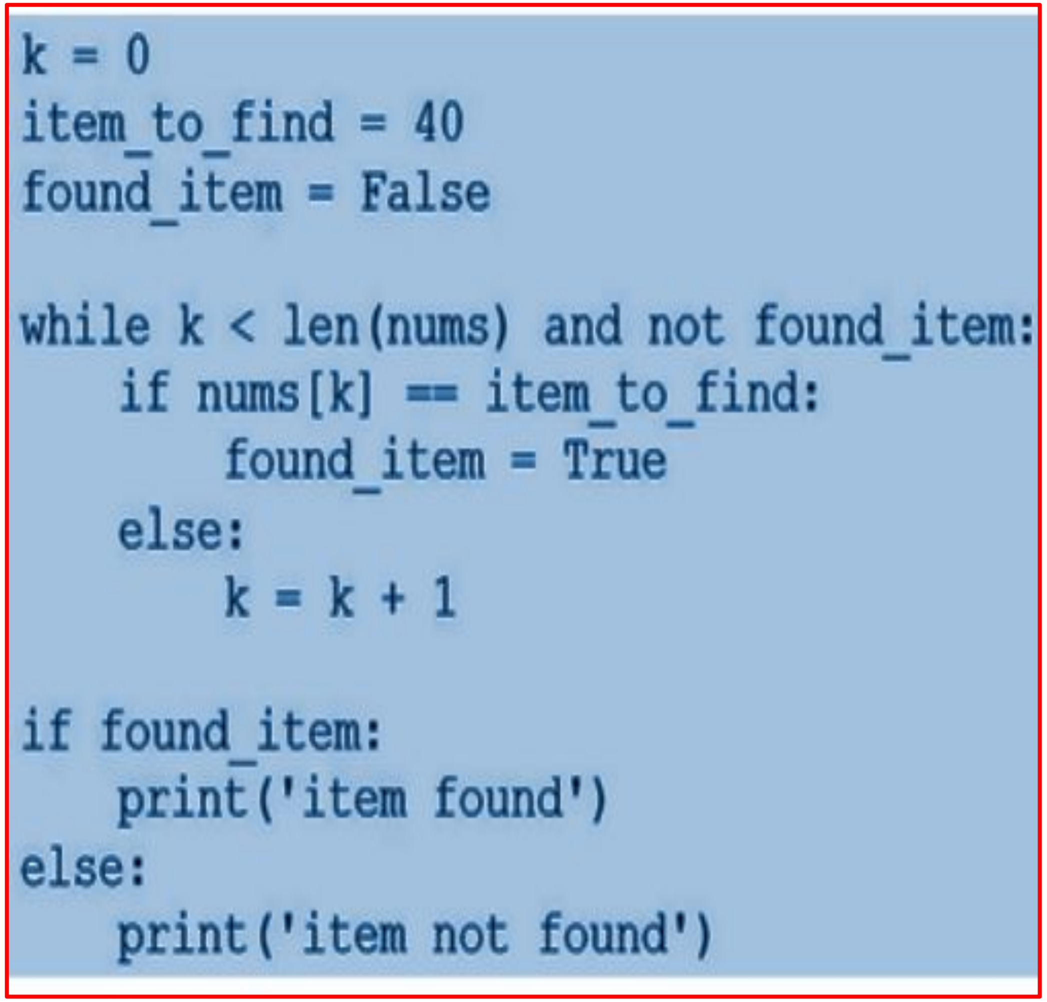
pCPS 4.3.4 while Loops and Lists (Sequences)

- There are situations in which a sequence is to be traversed while a given condition is **True**.

- In such cases, a while loop is the appropriate control structure.

- Another approach for the partial traversal of a sequence is by use of a for loop containing **break** statements.

- We **avoid** the use of **break** statements, favoring the more structured while loop approach.



```
k = 0
item_to_find = 40
found_item = False

while k < len(nums) and not found_item:
    if nums[k] == item_to_find:
        found_item = True
    else:
        k = k + 1

if found_item:
    print('item found')
else:
    print('item not found')
```

pCPS 4.4 More on python Lists

- We take a closer look at the **assignment** of **lists**. We also introduce a **useful** and **convenient** means of **generating lists** that the range function cannot produce, called **list comprehensions**

pCPS 4.4.1 Assigning and Copying Lists

- Because of the way that lists are represented in python, when a variable is assigned to another variable holding a list1, list2= list1, each variable ends up referring to the **same instance** of the **list** in **memory**.

- This has important **implications**.

- This issue **does not** apply to **strings** and **tuples**, since they are **immutable** and therefore cannot be modified

- When **needed**, a **copy** of a list can be made as given **list2 = list(list1)**

- When copying lists that have sublists, another means of copying, called **deep copy** , may be needed

- A **shallow copy** constructs a new compound object and then, to the extent possible inserts references into it to the objects found in the original.

- A **deep copy** constructs a new compound object and then, recursively, inserts copies into it of the objects found in the original.

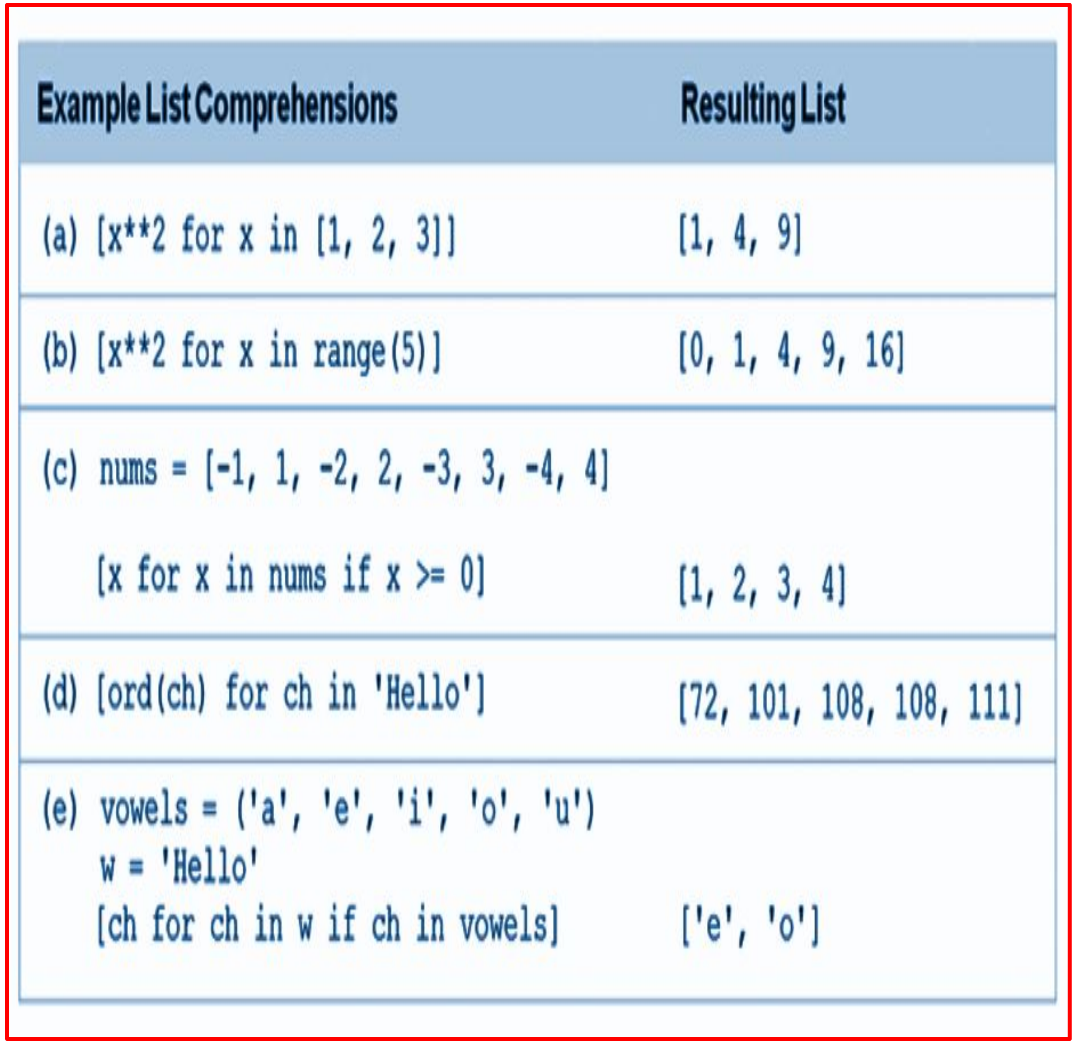
- The **range** function allows for the generation of sequences of integers in **fixed increments**.

- **List comprehensions** in python can be used to generate more **varied** sequences.

pCPS 4.4 List Comprehensions

In the **figure**,

- (a) generates a list of squares of the integers in list [1, 2, 3].
- In (b), squares are generated for each value in range(5).
- In (c), only positive elements of list nums are included in the resulting list.
- In (d), a list containing the character encoding values in the string 'Hello' is created.
- In (e), tuple vowels is used for generating a list containing only the vowels in string w.
- **List comprehensions** are a very **powerful feature** of **python**.



Example List Comprehensions	Resulting List
(a) `[x**2 for x in [1, 2, 3]]`	`[1, 4, 9]`
(b) `[x**2 for x in range(5)]`	`[0, 1, 4, 9, 16]`
(c) `nums = [-1, 1, -2, 2, -3, 3, -4, 4]` `[x for x in nums if x >= 0]`	`[1, 2, 3, 4]`
(d) `[ord(ch) for ch in 'Hello']`	`[72, 101, 108, 108, 111]`

(e) `vowels = ('a', 'e', 'i', 'o', 'u')`
`w = 'Hello'`
`[ch for ch in w if ch in vowels]`
	`['e', 'o']`

THANK YOU

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University
nitin.pujari@pes.edu

For Course Digital Deliverables visit www.pesuacademy.com